Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 1

Learning and generalisation in feed-forward networks — from perceptron learning to backprop

Nicolas Oulianitski Essipova

23rd January 2020

1 Main objectives and scope of the assignment

Our major goals in the assignment were

- to design and apply neural networks from scratch for the purpose of classification, function approximation, and generalization tasks.
- to identify key limitations of single-layered networks
- to configure and monitor the behavior of learning algorithms utilized in single- and multi-layer perceptron networks
- recognize risks associated with backpropagation and minimize them for robust learning of multi-layer perceptrons

The scope of this lab is that we will implement single- and multi-layer perceptrons from scratch, with focus on the associated learning algorithms, i.e: perceptron learning and generalized delta rule learning. Thereafter we will study their properties by means of simulations.

2 Methods

Python was used for this lab within a Jupyter Notebook environment running on Google Cloud Platform. NumPy was used for efficient numerical computations in higher dimensions along with PyPlot for visualizations.

3 Results and discussion - Part I

3.1 Classification with a single-layer perceptron

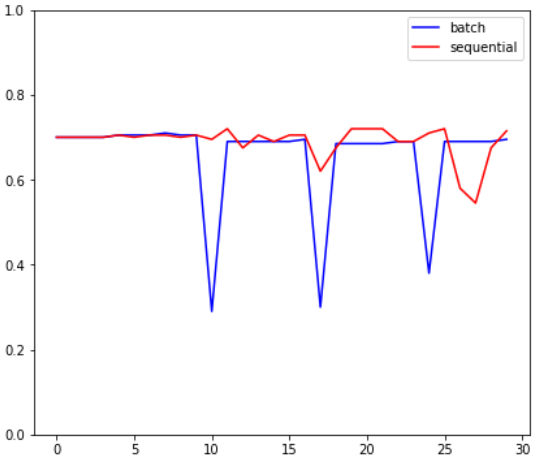
Isotropic Gaussian blobs were generated with 100 points for each class. Due to the necessary property of the weights being heterogenous to ensure that the learning algorithms would work, the weights for the perceptron were generated using NumPy's random generator multiplied with 0.01.

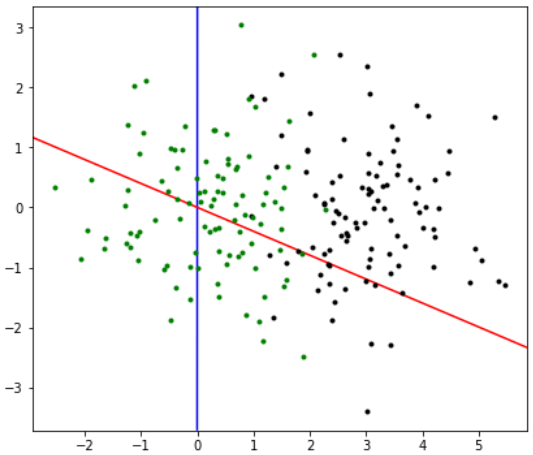
Since a perceptron is effectively trying to fit a set of datapoints through linear transformations by separating a hyperplane between them in a R^n space, wherein transfer functions are utilized as abstracting the output before feeding into the next layer of linear transformations; a bias is necessary so that the separation is not forced to go through the origin, i.e. [0,0] in our R^2 dimensional simulation.

Within the first simulation with using perceptron learning for both batch- and sequential learning for linearly separable data, it was observed that the end results were practically the same as both learning algorithms converged to separating the two categories of data. Most notable observations were that the sequential learning algorithm converged significantly faster than the batch learning algorithm, as it took less than 5 epochs across all tested learning rates for the former to converge, whereas the latter required 14 epochs for both learning rate set to 1.0, 0.1, and 0.01. At learning rate set to 0.001, it required the batch perceptron one additional epoch to converge.

Regarding the usage of generalized delta rule, it was observed that the batch mode had significantly slower convergence in contrast to sequential learning. Fastest convergence was observed with sequential learning using 0.01 learning rate.

Following phenomenon occurred when the perceptron was trained with a bias. As previously mentioned, a bias is necessary to ensure that the hyperplane boundary is not forced to go through the origin point (in this case, a line in two-dimensional space). The case of when no bias is needed in the perceptron to converge and correctly classify all data would be if the categories of classes can be linearly separated with a hyperplane (or in this case, a line) that passes through the origin point.





Finally, in regard to the perceptron performance on non-linearly separable data, it was observed that the network continuously kept looking for a way to converge but never managed as it is mathematically impossible for it to do so with the perceptron learning algorithm.

During use of delta learning algorithm, the network managed to converge more frequently at lower learning rates (<0.001) but frequently failed to do so at higher learning rates, and almost always failed at learning rates above 0.01.

Further experimentation was conducted on lower learning rates, and it was observed that the perceptron using generalized delta rule learning frequently converged to about 80% of the data being correctly classified.

With further experimentation using subsets of the dataset, it was observed that the training set accuracy became better as the subset sizes became unequal, but validation accuracy greatly suffered and thus generalization got worse. Conversely, best generalization was observed when both categories were represented equally in the subset. Thus, signifying the importance of data distribution for generalization.

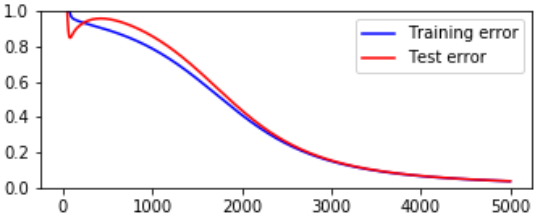
3.2 Classification and regression with a two-layer perceptron

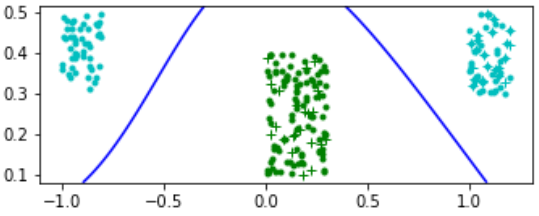
3.2.1 Classification of linearly non-separable data

It seems that one (decision boundary) or two plots (including learning curves) should suffice. Build a story around the questions in the assignment. Include concise motivation for your findings and potential interpretations/speculations.

The dataset was trained on nodes that scaled with the base of two, ranging h = 2^0 to 2^6, where h is the number of hidden nodes. It was observed that three out of the seven networks failed to converge within 5000 epochs, notably [2^0, 2^1, and 2^6]. The former two seemed to require more epochs as the tangent of both training and test error was in the right direction, whereas the graph for the latter first had its graph entirely missing any indications of progress. Both training error and test error started to be adjusted around epoch 2000 for h = 2^5, which was very peculiar.

Best performance with lowest mean squared error was observed by the network with 2^4 hidden nodes, with the following plot and decision boundary shown below. Mean squared error was falling from 2^0 to 2^4, but then started going up from 2^5 – possible indication of the network learning the noise from the training set and thus worsening its generalization towards validation set.





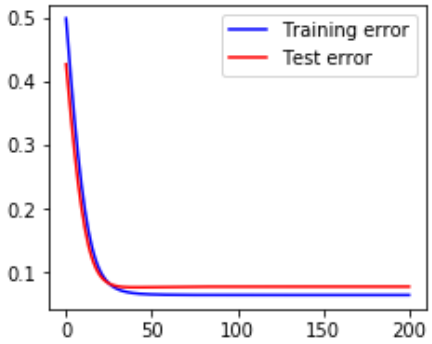
3.2.2 The encoder problem

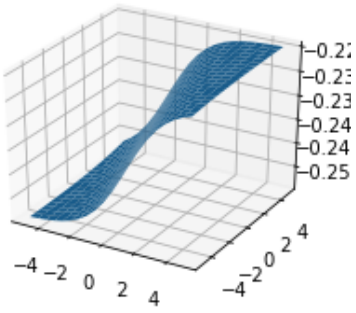
The autoencoder did not converge consistently but we did note improvements when increasing our learning rate. Reducing the number of hidden nodes from 3 to 2 resulted in the network never being able to create a learned representation of the input. Autoencoders can be used for data compression, dimensionality reduction, and data restoration.

3.2.3 Function approximation

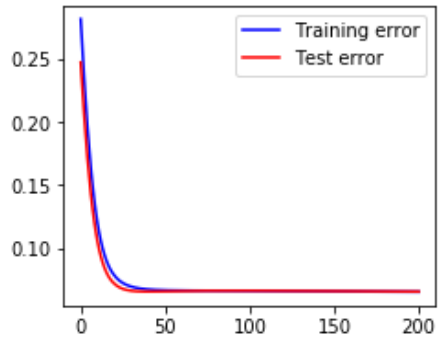
Mean squared error was used for evaluation, and the following were observed for the various hidden node sizes: h = 1, 2, and 25 in respective order.

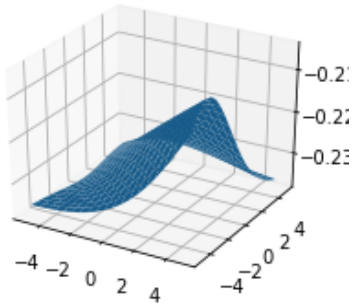
Hidden nodes = 1:



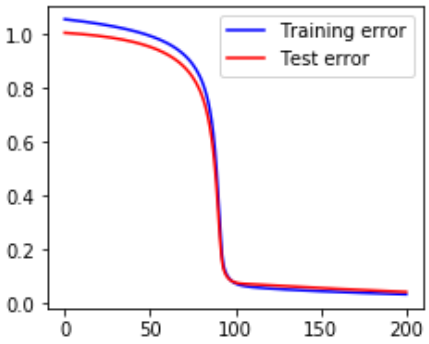


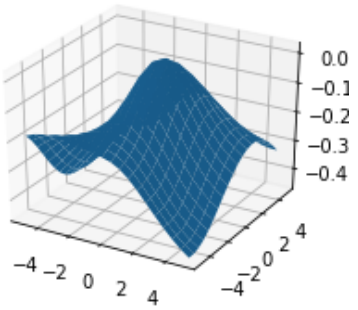
Hidden nodes = 2:





Hidden nodes = 25





The network displayed odd boundaries from hidden drones equaling 3 to 16. From 20 and onwards, it began to look more similar to what is shown with hidden nodes equal 25.

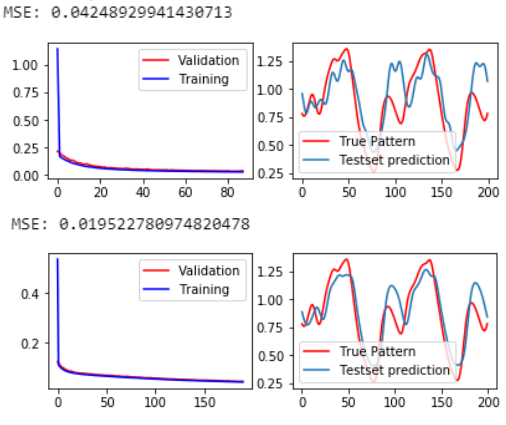
Reasoning behind these is that network with one hidden node was akin to a step function. Ranging upwards until 25 nodes, the network struggled to find a decision boundary that aptly generalized to the function. From 25 and onwards, the function began to look increasingly like the function itself.

Following is the table of the test mean squared error for the percentages of the training- and test set. As expected, results tend to be better as the model has more training data to train on. However, an interesting dip was observed when training data equated to 40% of the data and may require further statistical analysis to decide on whether it is statistically significant or not. It was also observed that convergence occurred sooner with less training data but at the cost of reduced generalization performance.

Training	Validation	Test MSE
0.8	0.2	0.0199
0.6	0.4	0.0332
0.4	0.6	0.0204
0.2	0.8	0.0615
0.1	0.9	0.0784

4 Results and discussion - Part II

We begin by generating and plotting our data from the Mackey-Glass time series function, and separating it into our train-, validation-, and test data. We separate it into 500, 500, and 200 data points respectively. Initially, we train a two-layer perceptron and look at the influence of hidden nodes and regularization on our network performance before moving onto investigating the same influences for a three-layer perceptron network. No conclusions will be made due to the small number of simulations, thus lacking significant statistical evidence to make any meaningful conclusions.



From the simulations that were run, it was difficult to ascertain whether regularization of L1 set to 0.01 had a statistically significant effect or not. In some cases, the model improved with certain number of nodes and in other cases it worsened. In the case of using three nodes, the mean squared error almost halved consistently with the regularized model (see images; unregularized model at the top and regularized at the bottom). However, setting the regularization too high, such as L1 set to 0.05, ended up in a consistently worse model due to increased bias.

When simulations were ran with a three-layer perceptron using the same regularization as the two-layer perceptron (L1 set to 0.01), and compared to the two-layer perceptron, it was noticeable that the three-layer perceptron consistently had higher mean squared error, which suggested that the two-layer perceptron had the better generalization capabilities to new data.

5 Final remarks

This lab was insightful with its demonstrations regarding practical aspects of implementing and utilizing a perceptron. It was interesting to observe how regularization impacts the generalization performance, the impact of which transfer function is used after each linear transformation, and how noise can be learned in the model.